INRIA

CENTRE DE RENNES
IRISA

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tel 954 90 20

Rapports de Recherche

N° 99

FORMALIZATION OF DLP (A DECENTRALIZED LOCKING PROTOCOL) USING ABSTRACT DATA TYPES

Serge M. MIRANDA

Novembre 1981

FORMALIZATION OF DLP (A DECENTRALIZED LOCKING PROTOCOL) USING ABSTRACT DATA TYPES

Serge M. MIRANDA

ABSTRACT

DLP is a fault-tolerant synchronization protocol which has been presented in an informal way[12] and its performances studied[13].

In this paper, we focus on an original formalization (specification, verification) of this protocol based on algebraically-defined data types.

FORMALISATION DE DLP (UN PROTOCOLE DE VERROUILLAGE REPARTI) A L'AIDE DES TYPES DE DONNEES ABSTRAITS

Serge M. MIRANDA

RESUME

DLP est un protocole de synchronisation tolérant aux fautes ("robuste") qui a été présenté d'une manière informelle dans [12] et dont les performances ont été évaluées dans [13].

Dans cet article nous nous concentrons sur une formalisation (spécification, vérification) originale basée sur l'approche algébrique des types de données abstraits pour prouver la correction de DLP.

FORMALIZATION OF DLP (A DECENTRALIZED LOCKING PROTOCOL) USING ABSTRACT DATA TYPES (1)

Serge M. MIRANDA

CERISS-INRIA. Université des Sciences Sociales
31042 TOULOUSE Cedex. FRANCE

Introduction

Synchronization issues in distributed data systems receive a good deal of attention in present literature. However various problems are still to be solved or at least explored in more details such as :

(i) - performance analysis.
(ii) - simulation
(iii) - fault-tolerance
(iv) - formalization (specification, validation, verification)

We presented[12] a synchronization protocol, named DLP, which ensures mutual consistency of distributed replicated data and which is fault-tolerant. This presentation was made is an informal way.

In this paper we explain in detail an original formalization of DLP based on abstract data types.

This article is divided into three parts:

- Section I recalls the basic features of DLP.
- Section II sums up the algebraic approach of abstract data types and its use as a formalization tool for synchronization protocols.
- Section III details the specification and verification of DLP using this formal model.

As a conclusion, we briefly tackle the validation issue in our model.

DLP presentation

After recalling the basic concepts of synchronization in distributed data bases, we briefly introduce DLP.

Synchronization concepts

A user of a data-base (centralized or distributed) may view or alter parts of that data base (DB) by means of transactions[6]. A transaction maps one consistent state of the DB to another. In a distributed data base a synchronization protocol has to be defined in order to avoid interference among remotely-initiated concurrent transactions[12].

Two important concepts emerged from the literature
- the concept of atomicity
- the concept of mutual integrity.

The concept of atomicity is attached to transactions ; transactions are said to be atomic if and only if concurrent transactions see all or none of the effects of every other transaction whatever the status of the distributed system.

The concept of mutual integrity ("consistency") is attached to data ; distributed replicated data are mutually consistent if every copy converges to the same state should update activity cease. We differentiate two cases of mutual integrity[12] : strong and weak mutual integrity. We say that a synchronization protocol ensures strong mutual integrity of replicated data if every "available" (potentially open to at least one retrieval access) copy is identical to the others. If two or more temporarily-different available versions of the same replicated entity may coexist in the distributed system, mutual integrity is said to be weak.

DLP summary

This paragraph sums up DLP presentation[12]. DLP is a decentralized locking protocol which was initially designed in a general-purpose-computer-network environment to ensure strong mutual integrity of replicated data even in case of any crash which may occur in the network (line crashes leading to autonomous subnetworks,...)[9]. From this type of network we infer the possibility of detecting remote crashes.

■■■ example : In the case of CYCLADES network, when a time-out attached to a message reception fires, a status-control message provided by the underlying virtual terminal protocol (D-STATUS message) is sent. The answer (MY-STATUS message) gives the status of the remote host. ■■■

Every time a DLP synchronization message is sent with an expected answer, a time-out is set.

At the reception of a user request (UPR message) also called "internal transaction", DLP is initiated. DLP encompasses two-steps of synchronization :

(i) - first step ; the idea is to broadcast the transaction, DBD message or "external transaction", to each controller involved in the session and wait for (N-1) positive acknowledgments - RDBD messages - (with N number of copies). This step corresponds to the initialization phase with temporary updates, security controls and concurrency-conflict resolution.

(ii) - second step ; when all RDBD messages are received ("unanimous consensus"), then the permanent update - DMS message - is broadcasted to every copy and there is another wait for (N-1) positive acknowledgments - RDMS message - which indicate the cor-

(1) This work is supported by an ADI contract (SIRIUS project) ## 80003.

Reprinted from FTCS-11, THE ELEVENTH ANNUAL INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, June 1981

rect local commitment. An end-of-session message (FDB) is then broadcasted to ensure a robust commitment. DLP is speed-independent since transactions can be initiated simultaneously at various sites. A transaction is temporarily rejected (during the first step of synchronization) when it has lower priority than the current transaction in progress. A transaction is permanently rejected when there exists a security violation in a given site. In both cases, an OINT message is used. The initiating site ("master controller") has control all the time over the progress of the internal transaction. During the first step of synchronization a "master" controller tries to establish its status. If it fails (there exists another master controller which initialized a higher-priority transaction), it broadcasts the OINT message to any site ("slave controller") which has already sent the RDBD message. The role of this OINT message becomes crucial in a partitionned data base (with no duplication) to solve concurrency conflicts[12]. The ground rule is whenever an object has been locked, it will only accept a message from its master controller (DMS or OINT) and ignore any other transaction (external or internal). This rule is necessary to avoid interference among conflicting transactions in a "partitionned" (not replicated) data base. A switch of status (from master to slave and vice versa) corresponds to :

- the reception of an internal update transaction with higher priority than the current one, in a slave controller which doesn't have its local copy locked.
- the reception of an external update transaction with higher priority than the current one, in the master controller which has not yet received (N-1) RDBD (performed a "global lock" on N copies).

The basic principles of DLP have been more or less proposed by other authors[1,18],.. However detailed description of functioning in case of concurrency conflicts or crashes, are lacking in most proposals indicated previously. We render DLP fault-tolerant ("robust") even in the case of a failure of the master controller (difference with[18]), which may occur during any broadcasting phase (some sites may receive the transmitted messages, others not).

Let us give the basic robustness characteristics of DLP. Every broadcasted message by the master controller includes a table, CTACT, which contains the identification of every active cooperating controller

The two major components of DLP fault tolerance are:

- a fully-replicated journal of modifications
- a virtual ring of resiliency.

Fully-replicated journal of modifications

The failure mechanism must record any copy modification sent to a site during a failure of that site so that the site can be brought up to date on recovery. Consequently a journal of modifications encompassing the list of transaction identifications not performed by the failed sites, is initiated for each slave controller which crashed during both steps. We discarded a decentralized journal system which could have led to an inconsistent state in the case of successive (bursty) failures. Therefore we choose a fully-replicated journal of modifications on each active site and we use the same synchronization session to ensure strong mutual consistency of this journal.

Virtual ring of resiliency

If a crash in a master controller occurs during the second step of synchronization, a controller among the active slave controllers must be elected "new master" in order to complete the synchronization session. In DLP, this election is made in a dynamic way through a virtual chaining of the slave controllers, which is performed at the beginning of the second step of synchronization. A chaining number (sending counter) is attached to every broadcasted message (and to the CTACT occurrences).

Failures during a broadcasting phase require the definition of two "forced" messages NSM and DFDB :

- NSM is the message broadcasted by a newly-elected master controller to the active slave controllers.
- DFDB is the message sent by a slave controller which is not eligible to become "new master", to the potential new-master controller after noting the failure of the current master controller.

Our mechanisms are compared with others[12].

Formal Model

The purpose of the following sections is to point out the abstract data type approach with its attractiveness as a uniform and rigorous framework for synchronization-protocol specification and verification.

Because of space limitations, it is not possible to detail :

(i) the algebraic approach of abstract data types (see[14,17] for this).

(ii) the formalization of a synchronization protocol (a complementary in-depth treatment can be found in companion papers[10,15])

Abstract data types (ADT's). In a first approximation we may say that an abstract data type or ADT represents an object with a familly of manipulation operators whose semantics are precisely defined.

We classified existing approaches to ADT specifications and point out the appeal of the algebraic method to complex-structure representation.[14]

We indicated also that this concept which appears in the realm of programming languages is in fact language independent and can be applied in the design of systems and the study of problems in a rigorous framework. We select the algebraic approach of ADT since structures involved in distributed systems are inherently complex.

Algebraic approach[4,7],..

We briefly recall the major features of the algebraic approach for abstract data types as presented in[4]

An algebra of one sort is roughly speaking a set of objects and a family of operators on the set. The set is called the CARRIER of the algebra.

Many-sorted algebras extend this notation by allowing the carrier of the algebra to consist of many disjoint sets. Each of these sets is said to have a SORT. The operators are sorted and typed but must be closed with respect to the carrier. An ADT can be defined as a MANY-SORTED ALGEBRA which is a family of sorts[3] with operations among them. The sort specifies the types used in the definition. Two basic kinds of sorts are involved in an ADT specification : the sort being defined and any number of sorts assumed previously defined in a similar fashion .

Operators of the algebra are indexed by pairs (w,s) where $w \in S^*$ (sort of the operands) and $s \in S$ (sort of the results). The symbol $\Sigma_{w,s}$ is used for the set of all operations with index (w,s) ; Σ is used for the union of all sets $\Sigma_{w,s}$ and is called the "signature" of the algebra.

A Σ-algebra is therefore determined by a triple ⟨S, Σ, ℰ⟩ where :

S is the set of "sorts", S : $\{s_1, s_2, .. s_i, ... s_n\}$ denoting the various types of objects which are required for that definition.

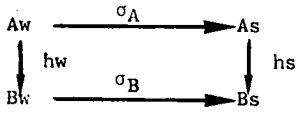
Σ is the set of operations Σ = {Σw,s} ,whose operands and results are objects making up the sorts in S (SYNTAX description).

ℰ is the set of equations which describe the properties of the Σ-operators and their relationships to one another (semantics description) ; each algebraic equation (or axiom) defines the results of various combinations of operators applied upon various operands.

Two many-sorted algebras are called Σ-algebras if they have the same signature. A very important concept is that of Σ-homomorphism which enables a functional linking between two Σ-algebras ;

Definition : given two Σ-algebras A and B, a Σ-homomorphism, h : A ⟶ B, is a family of functions ⟨hs : As ⟶ Bs, s ∈ S⟩ mapping each carrier in A (As) into the corresponding carrier in B (Bs) while preserving the operations, i.e.

∀σ ∈ Σw,s the diagram



commutes.

We use the Σ-homomorphism concept :

(i) to find the "best" algebra (up to isomorphism) in a family of algebras having the same presentation (S, Σ).

(ii) to map the abstract algebra into the implementation one defined as a Σ-algebra

(iii) to express layered abstractions between the distributed data base and the underlying transmission facility on the one hand ; between the distributed data base and the local DBMS on the other hand.

(iv) to represent parallelism among remote cooperating primitives.

The following sections illustrate points (iii)/(iv).

OBJ-O

We shall use a formalism close to OBJ-O[4,5,20,21] which represents one of the basic languages encompassing algebraically-specified data types.

At the time we had to select such a language,OBJ-O was the only one available (to our knowledge).Guttag's system[7] for symbolic execution of ADT's seems to suffer some inadequacies mainly at the syntactic level[21]. Today other powerful languages like AFFIRM[3], HISP[17],... have been proposed with very attractive and equivalent characteristics. OBJ-O in his own is still evoluting to OBJ-1[24] with the inclusion of the procedure feature,...

In OBJ-O, an algebra is a 4-tuple :

⟨ SORTS, OPS, VARS, SPECS ⟩ where

SORTS, OPS, SPECS correspond respectively to S, Σ, ℰ and VARS includes the definition of the working variables used in the axioms.

The error-operators ("ERROR OPS") and their semantics ("ERROR-SPECS") may be naturally defined in this language ; an extensive discussion of "error algebras" is presented[4].

The general syntax of an operator is given by :

$$\underbrace{OP}_{\text{operator name}} \; \underbrace{: S_1, S_2, ... S_N}_{\text{operand sorts}} \longrightarrow \underbrace{S_p}_{\text{result sort}}$$

(N : arity of the operator OP)

Prefix, infix, postfix, distributed-fix declarations are possible in OBJ-O.

Each sort has an equality relation which is built-in with syntax : $- : = - : S, S \longrightarrow S$

Hidden operators may be declared with the key-word "HIDDEN" placed after the result sort.

We use conditional equations (IF-THEN-ELSE) in the algebraic specifications ; a formal theory of conditional equations in ADT's is developped[22].

Formalization of a synchronization protocol

In order to formalize a synchronization protocol, we first specify any cooperating controller CT_i (process in charge of managing local data on site i) as an ADT named $SYNC_i$ and then express the functional parallelism between remote operators through the use of Σ-homomorphism between $SYNC_i$.

A controller specification is achieved in the following way :

- definition of a "global virtual object"
- definition of a reduced set of operators which apply to the global object and which express the synchronization-protocol semantics.

Global virtual object

The global virtual object consists of each local object semantically tied during a manipulation. A replicated object may be seen as a logical entity (the global virtual object) physically stored in several remote sites. This global object is characterized by a set of (global) states which represent the level of synchronization attained by the local controller during an update session.

Such an approach is consistent with the formal model we use ; as a matter of fact, one understanding of algebraically-specified ADT's is to see them as a set of internal states of a functional unit along with a system of basic functions enabling state transition and evaluation[19]. Such an ADT is called a "state ADT".

The state notion enables us

(i) to simplify the description of the different steps of a synchronization protocol (simplification of operator semantics,...)

(ii) to simplify the process of validation and verification by decomposing the protocol definition in clear steps.

(iii) to define clear re-entry points in a protocol for recovery purposes.

(iv) to express functional parallelism among remote operators by means of adequate homomorphism between states generated by these operators.

The number of significant states depends on the complexity and the degree of coordination defined in the synchronization protocol.

From the definition of mutual integrity two states appear to be important :

- the available state (A) associated with the local (replicated) entity which can be read.
- the unstable state (U) associated with the local entity being updated and which cannot be read.

Manipulation operators

Synchronization techniques in distributed systems are based on message transmission. By defining a certain number of primitives attached to synchronization message receptions, we render possible the implementation of a message-base system which has all the functionality of a data-abstraction-based system. Our approach can be extended to integrate robustness i.e. the degraded functioning of the protocol in case of any failure in the distributed system (line, site,...).We control the crash consequences in a synchronization session by specifying the effects of a crash. In order to do so, we define the semantics of control-message receptions indicating the crash of a remote cooperating controller. These control messages are managed by an underlying network protocol (VTP in the case of CYCLADES network,...).

The study of synchronization protocol leads us to consider three basic operators for an update session :

- the operator of global preparation : PREPAREG
- the operator of global manipulation: MANIPULEG
- the operator of global termination : TERMINEG

PREPAREG

The operator of global preparation corresponds to:

- on one hand, the local initialization of the update session which includes all the actions performed by a given controller to start a global session: time-stamp acquisition[23], new value computation[23], ticket allocation[8], priority definition[1,12],...
- on the other hand, the global initialization which includes the strategy to solve concurrency conflicts : first revolution in the virtual ring of transmission[1], transaction sequencing using tickets or priorities[8,12],....

Once the global consensus attained on a given transaction,either unanimous[1,8,12],.. or majoritary[23],. the MANIPULEG operator is invoked.

MANIPULEG

The operator of global manipulation includes the transaction processing (permanent update) on each copy.

TERMINEG

The operator of global termination is associated with either a controlled commitment (protocols ensuring strong mutual integrity) or a robust commitment (two-step commit protocols,...) which may include a transaction roll-back.

Each of these operators :

- generally corresponds to the processing of the same synchronization message by every controller involved in the update session.
- is atomic ; as a consequence the order of actions associated with an operator is not significant.
- maps an input global state and a set of input messages into an output state and a set of output messages.

Any sequence of SYNC operators is generally reduced to the sequence PREPAREG-MANIPULEG-TERMINEG called "complete" canonical representation.

Some operators in this sequence may be omitted :

- PREPAREG in some controllers for protocols ensuring weak mutual integrity (possibility of a direct transition in the modification step).
- MANIPULEG in the case of transaction rejection ; the canonical representation is then said to be "partial".
- TERMINEG in the case of inclusion in MANIPULEG.

Specification of functional parallelism

The concept of Σ-homomorphism allows the expression of functional links between Σ-algebras defined :

- either at the same abstractional level to find the "best" Σ-algebra (initial algebra)in a family of Σ-algebras sharing the same presentation (S,Σ).
- or at different abstractional levels.

In this remark, we did not make any assumption regarding the distribution of entities or algebras. As a matter of fact, the same abstractional level may incorporate distributed entities and operators. It is the case when taking into account Σ-algebras defined at the same abstractional level, namely the global-integrity level, which involves distributed data management.

Consequently we shall express parallelism semantics between controllers, with Σ-homomorphism among $SYNC_i$ algebras. A morphism between global states represents a functional correspondance and not necessarily a simultaneity between operators generating these states. Such a morphism is attached to a synchronization message (emitted when generating the corresponding global state). A message can be lost. However we assume, as most proposals, that a message is always delivered : the loss can be detected and the message transmitted again (as in CYCLADES network,.)

The specification of a controller includes the writing of all the actions associated with the reception of any synchronization message.

In order to do so we have to give an explicit form to :

- the various states that may be taken by a global entity on a given controller.
- the functional links, symbolized with the Σ-homomorphism, which exist among remote states. These links represent the semantics of the synchronization protocol.

As a summary, a synchronization protocol is formally described by :

- the specification of a controller using an object-oriented language like OBJ-0.
- the definition of a Σ-homomorphism between $SYNC_i$

We formally specify[10,11,15] these three basic operators for four protocols which are representative of a particular class of solution[1,8,23,25] in a fail safe environment. This approach is extended here to formalize robustness in the case of DLP.

From our formalism we can infer simple proofs of verification. We develop[11] a formal framework for verification of a synchronization protocol specified with algebraically-defined data types. This verification relies on theorems which indicate the type of mutual integrity achieved by a protocol formally specified in our model.

The proof of the following theorem is presented[11] and will be published in a companion paper.

Theorem of strong mutual integrity

Let $SYNC_m$ be the ADT which represents the functioning of the m^{th} controller.

Let S be a system of conflicting transactions and T_i and T_j, be two transactions of S.

A synchronization protocol formally specified with algebraically-defined data types, ensures strong mutual integrity of replicated data IF :

(i)the sequence of operators of a $SYNC_m$ is reduced to a canonical representation and T_i,T_j are not "interleaved" in the update phase (the subsequences MANIPULEG-TERMINEG are mutually exclusive)

(ii) there exists a "total" Σ-homomorphism among $SYNC_m$ on states generated by the operators of the canonical representation for a given transaction ("total" refers to every controller involved in the synchronization session).

DLP formalization

A controller is formalized by a Σ-algebra SYNC. The operators of the SYNC algebra map a global state of the local entity-copy (upon the reception of a given synchronization message) into another state.

A global state (GES) is a triplet ⟨P, CS, CTS⟩ where

- P : Priority of the corresponding transaction T_i
- CS : Copy state which can be "free" (L), "occupied" (O) or "blocked" (B), depending on whether there is: (i) no update session (ii) update session initialized where the master status can still switch for concurrency reasons (iii) an update session in the commitment phase.
- CTS: Controller state which can be slave (SL) or master (MT).

In the specifications of the SYNC algebra we shall use auxiliary types whose signatures are indicated below.

OBJECT	OPERATORS	COMMENTS
Message (M)	TRANSMIT(M,P,I, $CTID_v$ ALL) RECEIVE(M,P, CTACT,Q,GES, I) WAIT (M,I)	ALL : message broadcasting to every controller whose identification is in CTACT. CTID: controller identification. CTACT: table of active controllers. P : priority I : message counter Q : waiting list of temporary rejected transactions
Local entity (EL)	PREPAREL (EL) MANIPULEL (EL) TERMINEL (EL)	These operators which correspond to locking ones are given an explicit form for a general-purpose DB[11]
Transaction (T)	PROCESS (T)	
Priority (P)	DEFINE (P,T_i)	(P,T_i) will be noted P_i
Queue (Q)	ENQ (T,Q) DEQ (Q) EMPTY ? (Q) ...	
Integer (INT)	TEST (A,B) SUP (A,B)	TEST (A,B):= IF A=B THEN TRUE ELSE FALSE SUP (A,B):= IF A>B THEN TRUE ELSE FALSE

....

DLP specifications

We are going to give the specifications of a controller using a language very close to OBJ-0. From these specifications we shall analyse state switchs and robustness integration.

SYNC specifications

```
OBJECT SYNC ⟨ith controller⟩
SORTS  GES,EL,INT,BOOL,T,M,Q(ueue),L(IST),P
OPS    PREPAREG (') : M x GES ──► GES   } Operators attached to the
       MANIPULEG ('): M x GES ──► GES   } fail-safe network case (*)
       TERMINEG (') : M x GES ──► GES   }
       ID           : GES ──────► GES   } operators attached to the
       ELECTNSM'    : M x GES ──► GES   } robust commitment.
       ATTFIN'      : GES ──────► GES   }
VARS   P     : INT ⟨priority⟩
       CT    : INT ⟨controller identification⟩
       CTMT  : INT ⟨controller identification for a
                    master controller⟩
       CTSL  : INT ⟨controller identification for a
                    slave controller⟩
       N-CTMT: INT ⟨newly elected CTMT⟩
       L     : CS ; O:CS ; B:CS ; λ = O_v B;
       SL    : CTS ; MT:CTS ;
       DBD   : M ; ⟨demand of distant blocking ;
                    temporary update;...⟩
       RDBD  : M ; ⟨DBD ACK after local locking, local security checking...⟩
       OINT  : M ; ⟨interrupt order ; DBD ACK in case of higher-priority concurrent
                    transaction or security violation⟩
       DMS   : M ; ⟨demand of synchronized update ;
                    permanent update⟩
       RDMS  : M ; ⟨DMS ACK after local update⟩
       FDB   : M ; ⟨end of global blocking ; end of
                    local log synchronization⟩
       NSM   : M ; ⟨new master-site message⟩
       SSD   : M ; ⟨down-slave-site message⟩
       MSD   : M ; ⟨down-master-site message⟩
       DFDB  : M ; ⟨demand of commitment to a new
                    master controller⟩
       C     : INT ⟨counter of received messages of
                    the same type⟩
       CHN   : INT ⟨chaining number ; enables the
                    creation of the virtual ring of
                    resiliency⟩
       CTACT : LIST⟨list of active controllers sent
                    during the broadcasting of any
                    synchronization messages from
                    the master controller in order
                    to synchronize log updates ;
                    logs are to be identical in each active controller⟩
       Q     : QUEUE⟨queue of temporarily rejected
                    transactions ; transactions may
                    be taken into account in the end
                    of the previous update synchronization session (matter of policy)⟩
       LOG(EL): QUEUE⟨queue of modifications attached to the local entity⟩
```

(*) In fact, the general functional definition of an operator is of the form $M \times GES \xrightarrow{OP} GES \times M$

SPECS

‹For a given synchronization session, the controller status is either master or slave. Therefore we separate their operator semantics›

‹operator semantics for a master controller›

‹UPR reception›

RECEIVE (UPR,-,CTACT,-;(-,L,SL),-) : = PREPAREG(UPR,(-,L,SL)) ; (1)

RECEIVE (UPR,-,CTACT,-;(P_k,0,MT),-) : = DEFINE (P_i) ;
IF SUP (P_i,P_k) = TRUE
THEN (ENQ (Q,(UPR,P_k)) ;
TERMINEG (-,(P_k,0,MT)) ;
PREPAREG (UPR,(P_i,L,SL)););
ELSE (ENQ (Q,(UPR,P_i)) ;
ID (P_k, 0, MT);)(*1);

RECEIVE (UPR,-,CTACT,-; (P_k,B,MT),-): = DEFINE (P_i) ;
ENQ(Q,(UPR,P_i)) ;
ID (P_k, B,MT) ;

‹DBD reception›

RECEIVE (DBD,P_i,CTACT,-,;(P_k,0,MT),)) : = IF SUP (P_i,P_k) = TRUE
THEN (SWITCHG(P_i,(P_k,0,MT));) ;
(4) ELSE (ID(P_k,0,MT)) ;

RECEIVE (DBD,P_i,CTACT,-,;(P_k,B,MT),-) : = ENQ(Q,(DBD,P_i)); ID(P_k,B,MT) ;

‹OINT reception corresponding to a security violation in a slave controller›

RECEIVE (OINT,P_k,-,-;(P_k,0,MT),C) : = TERMINEG (-,(P_k,0,MT)) ;
TRANSMIT (OINT,P_k,-,ALL)) ;

‹RDBD reception›

RECEIVE (RDBD,P_k,-,-;(P_k,0,MT),C) : = INCR(RDBD,C) ;
IF TEST (C,N-1) = TRUE
(2) THEN (MANIPULEG((N-1)RDBD,(P_k,0,MT))-;) ;
ELSE (WAIT(RDBD,N-1-C) ;
ID (P_k,0,MT);) ;

‹RDMS reception›

RECEIVE (RDMS(P_k,-,-;(P_k,B,MT),C) : = INCR(RDMS,C) ;
IF TEST (C,N-1) = TRUE
THEN (TERMINEG((N-1)RDMS,(P_k,B,MT))) ;
(3) ELSE (WAIT(RDMS,N-1-C) ;
ID (P_k,B,MT);) ;

‹SSD reception›

RECEIVE (SSD,-,-,-; (P_k, λ ,MT),C) : = ‹CTACT update›
DECR (N) ;
IF TEST (C,N-1) = TRUE
THEN IF λ = "0"
THEN(MANIPULEG((N-1)RDBD,(P_k,0,MT))) ;
E SE (TERMINEG((N-1)RDMS,(P_k,B,MT))) ;
ELSE (WAIT(M,N-1-C) ;
ID (P_k, λ ,MT);)

‹NSM reception›

RECEIVE (NSM,P_i,CTACT,-;(P_k,0,MT),-) : = IF‹(T,P_i) IN LOG (EL)›
‹update already performed›
THEN (TERMINEG'(DMS,(P_i,-,-));) ;
ELSE (ENQ(Q,(UPR,P_k));
SWITCHG (P_i,(P_k,0,MT)) ;
MANIPULEG'(NSM,(P_i,B,SL));) ;

‹DFDB reception›

RECEIVE (DFDB,P_i,-,-; (P_k,0,MT),-) : = TRANSMIT (FDB,P_i,1,(CTSL,DBD,P_i))(*2) ;
ID (P_k,0,MT) ;

PREPAREG (UPR,(-,L,SL)) : = ID(0) ; SWITCH(SL) ;‹if priority not defined,do DEFINE (P_i)›
ID(P_i,0,MT);

(*1) The identity operator applied to GES,C S,CTS will be noted ID (x) with x, output of the operator.
(*2) CTID is of the type "identification of the controller managing the message DBD having priority P_i"

```
          (5)                PREPAREL (EL) ;    ⟨local locking ; security checking,...⟩
                             IF⟨no security violation⟩
                             THEN (TRANSMIT (DBD,P_i,-,ALL) ;
                                 C=0 ; WAIT (RDBD,N-1) v WAIT(SSD v NSM v DFDB v OINT,-);) ;
                             ELSE(TERMINEG(-,(P_i,0,MT)));) ;

MANIPULEG ((N-1)RDBD,(P_i,0,MT)) : = ID(B) ;
                             ID(P_i,B,MT) ;
          (6)                MANIPULEL(EL) ⟨PROCESS(T),..... ; ⟩
                             ENQ(LOG(EL),(T,P_i)) ;
                             TRANSMIT (DMS,P_i,-,ALL) ;
                             C=0 ; WAIT(RDMS,N-1) v WAIT(SSD,-) ;
TERMINEG ((N-1) RDMS;(P_i,B,MT)) : = TRANSMIT (FDB,P_i,-,ALL);TRANSMIT(RUPR,-,1,-) ;
                             ID(L); ID(SL) ;
          (7)                ID(-,L,SL) ; DEQ(Q) ....
                             TERMINEL (EL) ;
SWITCHG (P_i,(P_k,0,MT))     : = TERMINEG(-,(P_k,0,MT)) ;
                             TRANSMIT (OINT,P_i,C,(CTSL,DBD,P_k)) ;
          (8)                PREPAREG' (P_i,L,SL) ;

⟨ operator_semantics_for_a_slave_controller ⟩

⟨ UPR reception ⟩

RECEIVE (UPR,-,CTACT,-;(P_k,0,SL),-)  : = DEFINE(P_i) ;
              IF SUP (P_i, P_k) = TRUE
              THEN (SWITCHG'(UPR,P_i,(P_k,0,SL)) ; )
              ELSE (ID(P_k,0,SL) ; ENQ(Q,(UPR,P_i)););
RECEIVE (UPR,-,CTACT,-;(P_k,B,SL))    : = ID(P_k,B,SL) ; DEFINE(P_i);)
                                          ENQ (Q,(UPR,P_i)) ;

⟨ DBD reception ⟩

RECEIVE (DBD,P_i,CTACT,-;(-,L,SL),-)  : = PREPAREG' (DBD,P_i,(-,L,SL)) ;
RECEIVE (DBD,P_i,CTACT,-;(P_k,0,SL),-) : = IF SUP (P_i,P_k) = TRUE
                                          THEN (TERMINEG'(-,(P_k,0,SL)) ;
                                                PREPAREG'(DBD,(P_i,L,SL)););
                                          ELSE (ID(P_k,0,SL););
RECEIVE (DBD,P_i,CTACT,-;(P_k,B,SL),-) : = ID(P_k,B,SL);ENQ(Q,(DBD,P_i));

⟨ DMS reception ⟩

RECEIVE (DMS,P_i,CTACT,-;(P_i,B,SL),-) : = MANIPULEG'(DMS,(P_i,B,SL)) ;
RECEIVE (RDBD,P_k,-,-;(P, λ,SL),-)     : = TRANSMIT (OINT,P,1,(CTSL,DBD,P_k)) ;
                                          ID(P, λ ,SL) ;

⟨ FDB reception ⟩

RECEIVE (FDB,P_i,CTACT,Q;(P_i,B,SL),-) : = TERMINEG'(FDB,(P_i,B,SL)) ;

⟨ OINT reception ⟩

RECEIVE (OINT,P_i,-,-; (P_k,0 v B,SL),-) : = IF TEST(P_i,P_k) = TRUE
                                          THEN (TERMINEG'(-,(P_i, 0 v B,SL))) ;
                                          ELSE (IF SUP (P_i,P_k)      = TRUE
                                                THEN (TERMINEG'(-,(P_k,0 v B,SL)) ;
                                                      PREPAREG'(-,(P_i,L,SL));) ;
                                                ELSE (ID(P_k,0 v B,SL)); ) ;

⟨ MSD reception ⟩
RECEIVE (MSD,-,-,-; (P_k,0,SL),-)      : = TERMINEG' (-,(P_k,0,SL)) ;
RECEIVE (MSD,-,-,-; (P_k,B,SL),-)      : = IF(T,P_k) in LOG(EL)⟩
                                          THEN (IF CHN = 1
                                                THEN (ELECTNSM'(MSD,(P_k,B,SL));) ;
                                                ELSE (ATTFIN' (MSD,(P_k,B,SL)););) ;
          (9)                             ELSE  (TERMINEG' (-,(P_k,B,SL));) :

⟨ NSM reception ⟩
RECEIVE (NSM,P_i,CTACT,-;(-,L,SL),-)   : = PREPAREG' (NSM,(P_i,L,SL)) ;
                                          MANIPULEG'(NSM,(P_i,B,SL)) ;
          (12)                           ⟨ a different session has been initialized ⟩
RECEIVE (NSM,P_i,CTACT,-;(P_k,0,SL))   : = TERMINEG'(NSM,(P_k,0,SL)) ;
          (13)                           PREPAREG'(NSM,(P_i,0,SL)) ;
                                          MANIPULEG'(NSM,( Pi,B,SL)) ;
```

```
RECEIVE (NSM,P_i,CTACT,-;(P_k,B,SL),-)        : =  IF TEST(P_i,P_k) = TRUE
                                                   THEN (MANIPULEG'(NSM,(P_i,B,SL));) ; ‹same session›
          (14)                                     ELSE (TERMINEG'(NSM,(P_k, B,SL)) ;‹different session›
                                                         PREPAREG'(NSM,(P_i,L,SL)) ;
                                                         MANIPULEG(NSM,(P_i,B,SL));) ;
‹DBDB reception by the slave controller which got the DMS vith the minimum CHN›
RECEIVE (DFDB,P_i,-,-;(-,L_v0,SL),-)          : =  TRANSMIT (FDB,P_i,1,(CTSL,DBD,P_i)) ; ID(-,L_v0,SL) ;
RECEIVE (DFDB,P_i,-,-;(P_k,vB,SL),-)          : =  IF TEST (P_i,P_k) = TRUE
                                                   THEN (ELECTNSM'(DFDB,(P_i,B,SL));) ; ‹CHN = 1›
                                                   ELSE (TRANSMIT(FDB,P_i,1,(CTSL,DBD,P_i)) ; ID(P_k,B,SL);) ;
PREPAREG' (DBD_vNSM,(P_i,L,SL))               : =  ID(0) ; ID(P_i,0,SL);PREPAREL (EL)
                                                   IF ‹ no security violation ›
                                                   THEN (ID(P_i,B,SL) ;
                                                   IF ‹ DBD messages ›
                                                   THEN (TRANSMIT(RDBD,P_i,1,(CTMT,DBD,P_i)) ;
                                                         WAIT(DMS_vNSM_vMSD_vOINT,-);) ;
                                                   ELSE (TRANSMIT(OINT,P_i,1;(CTMT,DBD,P_i)) ;
                                                         TERMINEG'(OINT,(P_i,0,SL)) ;);
MANIPULEG'(DMS_vNSM,(P_i,B,SL))               : =  ID(P_i,B,SL);MANIPULEL(EL);
                                                   IF(T,P_i) ‹ not in LOG (EL)›
          (15)                                     THEN (PROCESS(T) ;...;
                                                  ‹CHN update, Q processing .....›
                                                   TRANSMIT(RDMS,P_i,1,(CTMT,DBD,P_i));
                                                   WAIT (FDB_vNSM_vMSD_vDFDB,-);) ;
TERMINEG' (FDB,(P_i,λ,SL))                    : =  TERMINEL (EL) ;
                                                  ‹Q processing›
                                                   ID(L) ; ID (-,L,SL) ;
SWITCHG' (UPR,P_k,(P_i,0,SL))                 : =  TERMINEG'(-,(P_i,0,SL));
                                                   PREPAREG (UPR,(P_k,L,SL)) ;
ELECTNSM' (MSD_vDFDB,(P_i,B,SL))              : =  DECR (N) ; SWITCH (SL) ; ID (P_i,B,MT) ;
                                                  ‹CTACT update›
          (10)                                     DECR (CHN) ; TRANSMIT (NSM,P_i,-,ALL) ;
                                                   WAIT (RDMS_vSSD,N-1) ;
ATTFIN' (MSD,(P_i,B,SL))                      : =  DECR (N) ; ID(P_i,B,SL) ; ‹CTACT update›
                                                   TRANSMIT (DFDB,P_i,1,( CTMT,DBD,P_i)) ;
          (11)                                     WAIT (NSM_vFDB,-);
```

TCEJB0

Notes 1 - Other message receptions may occur; we did not represent them since either they do not affect GES or they correspond to errors.

2 - We may give an example of "wait" semantics with the VTP of CYCLADES network.

```
WAIT (M,-) : = IF TIME-OUT  = TRUE
               THEN TRANSMIT (D-STATUS,-,-,-) ; WAIT (R-STATUS) ;

RECEIVE (R-STATUS) -,-,-,-,-) : = IF (R-STATUS)  = 'UP'
                                  THEN TRANSMIT (M',-,-,-) ‹retransmission›
                                  ELSE
                                       IF (R-STATUS) = "NOT READY"
                                       THEN ‹wait for end of recovery procedure›
                                       ELSE ‹CTACT update›    ; N = N-1 ;
                                            ‹R-STATUS = "OFF"›;
```

Analysis of state switchs

These specifications make unambiguous the local or global object-state switch ; for example, let us consider the global object attached to a master controller ; we get :

L ⟶ 0 when the CT receives an UPR ; this switch is associated with PREPAREG whose definition in terms of received message is expressed in (1) and semantics are presented in (5)

0 ⟶ B when the CT-MT receives (N-1) RDBD with N number of active cooperating controllers ; indicated in MANIPULEG definition (2) and specified in (6)

B ⟶ L(') when the CT-MT receives (N-1) RDMS ; definition with TERMINEG (3) and (7)

We can make the same analysis for the controller-status switch ; i.e. the switch from master to slave during the first step of synchronization (associated with global-object state 0) is defined in(4) and its semantics are introduced in (8) ; from (4) we infer that this switch occurs when a higher-priority DBD is received during the first step of synchronization ; in (8) we notice that this switch generates a broadcasting of C x OINT messages (C being the counter of received RDBD).

The same clear analysis can be conducted for a local-object-state switch attached to a slave controller whose basic operators are PREPAREG',MANIPULEG' and TERMINEG'.

We can note that on a master controller, the global -object-state can be 0 (and still switch) while some of the local-object-states can be B ; this corresponds to the partial processing of DBD messages.

This entails the requirement of OINT messages sent by the (old) master controller when a MT⟶SL switch occurs. We chose this approach based on the locus of control in the site where the UPR originated in order to be able to extend naturally DLP to the update synchronization of partitionned data bases.

Analysis of robustness specifications

When taking into account the robust commitment we introduce two operators performed by a slave controller :

1 - ELECTNMS' corresponding to the slave controller which noticed the failure of the current master and gets the smallest CHN number (new potential MC).

2 - ATTFIN' corresponding to a slave controller which noticed the failure of the current master,did not get the smallest CHN number, sends DFDB to the new potential MC, and waits for the NSM or FDB message.

It is important to note that a NSM message is a forced message which must be prooessed ; it may be received by :

(i) a slave controller in its first step of synchronization which either waits for the DMS message or gave up the synchronization session (and eventually initiated a different session). This latter situation may arise when a slave controller noticed the failure of the master controller while waiting for the DMS message and when the master controller failed during the DMS-broadcasting phase, the local state is undefined (-, λ ,-).

(ii) a slave controller in its second step of synchronization which either processing a DMS message, or waiting for the FDB message,(or the NSM itself after noting the master-controller failure) ; the local state is (-,B,SL).

In both cases the NSM is acknowledged by a RDMS message (transmitted in MANIPULEG').

DLP verification

We can express parallelism between two remote-controller ADT's, $SYNC_i$ and $SYNC_j$.

In order to do so we present a functional coupling (ψ) between the states of an entity on a master and slave controller. This functional coupling represents the transmission of a synchronization message. We separate three cases :

- reliable environment with no concurrency conflicts.
- reliable environment with concurrency conflicts.
- unreliable environment.

(i) reliable environment with no concurrency conflicts.

Paired states indicated below are said to be "homogeneous" from a synchronization point of view.

Master controller (MC)		Slave controller (SC)	
$(-,L,SL)$	$\xrightarrow{\psi}$	$(-,L,SL)$	‹before modification›
$(P_i,0,MT)$	$\xrightarrow{\psi(DBD)}$	$(P_i,0,SL)$	$\vee$ (P_i,B,SL)
$(P_i,0,MT)$	$\xrightarrow{\psi(OINT)}$	$(P_i,0\vee B,SL)$	‹security violation›
(P_i,B,MT)	$\xrightarrow{\psi(DMS)}$	(P_i,B,SL)	‹update phase›
$(-,L',SL)$	$\xrightarrow{\psi}$	$(-,L',SL)$	‹new version after modification›

(ii) reliable environment with concurrency conflicts

Paired states are said to be "compatible" from a synchronization point of view.

$$\underset{MC}{(P_i,0,MT)} \xrightarrow{\psi(DBD)} \underset{SC}{(P_j,0,MT)} \qquad (1)$$

$$\underset{MC}{(P_i,0,MT)} \xrightarrow{\psi(OINT)} \underset{SC}{(P_i,B,SL)}$$

(iii) unreliable environment

$$\underset{MC}{\text{CRASH}\{MCD\}} \xrightarrow{\psi(MSD)} \underset{SC}{(P_i,0\vee B,SL)}$$

$$\underset{SC}{(P_i,B,MT)} \xrightarrow{\psi(NSM)} (P_i,B,SL)\vee(-,0,-)$$

$$\underset{SC}{(P_i,B,SL)} \xrightarrow{\psi(DFDB)} (P_i,S,SL)\vee(-,0,-)\vee(P_i,B,MT)$$

These states are said to be "robust" from a synchronization point of view.

Assertion 1

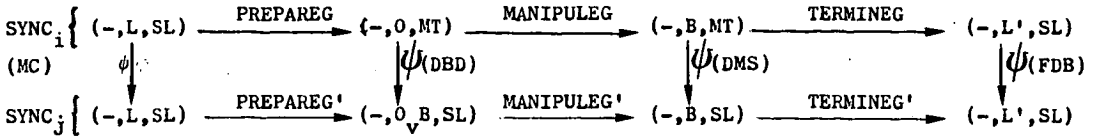
DLP ensures strong mutual integrity.

Proof

We shall verify the theorem of strong mutual integrity in every environment case.

(i) reliable environment with no conflicts

The theorem verification is straight-forward ;from DLP specifications (for a MC and a SC) and the functional coupling among homogeneous states,we can build the following diagram commutations which visualize the verification of the theorem.



Notes

1 - We visualized a global synchronization session with a MC and any SC.

2 - The canonical representation may be written TERMINE(MANIPULE(PREPARE(DBD,L_v0),DMS,B),FDB,L').

3 - In case of security violation, mutual integrity is achieved. The canonical representation is TERMINE (PREPARE(DBD,L_v0),OINT,L).

(ii) reliable environment with conflicts

In order to verify the theorem of strong mutual integrity, it is sufficient to consider two conflicting transactions T_i and T_j since there exists a total ordering on transactions. The proof is reduced to show that starting from "compatible" states we reach "homogeneous" states after a finite number of transitions. We have two possible (and symmetric) situations depending on the choice of the priority.

Let us assume we have $P_k < P_i$.

The most general situation is depicted in figure 1

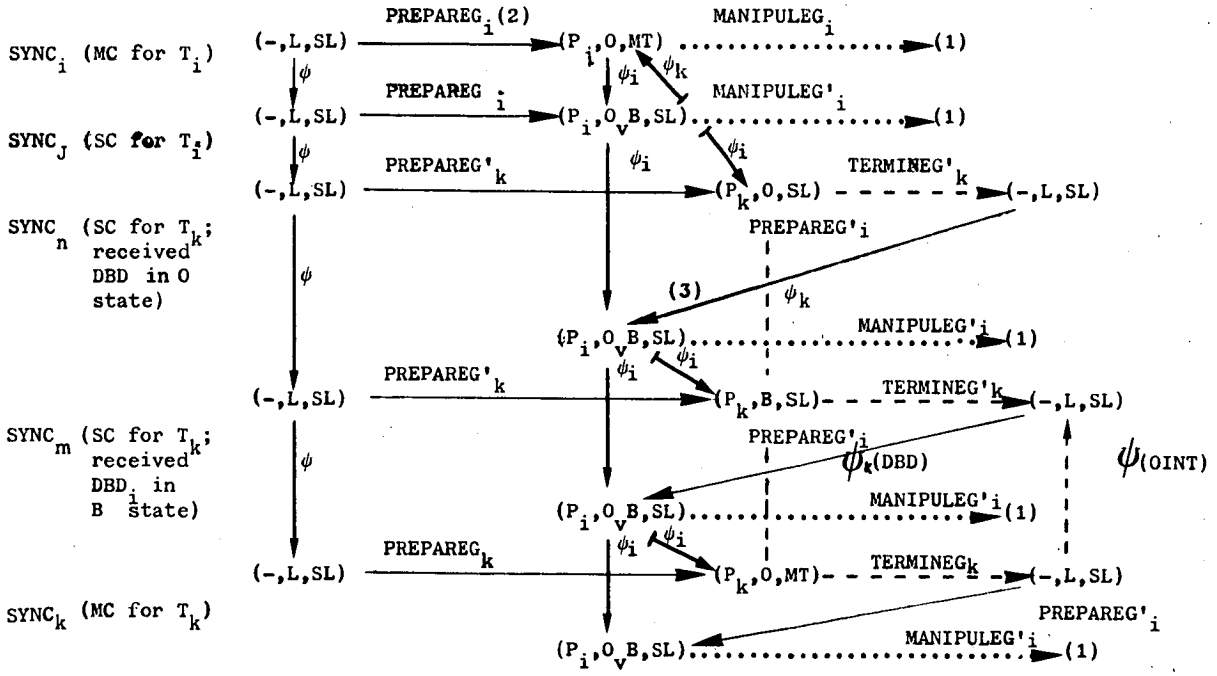


Figure 1 : Commutation diagram associated with a reliable environment in a conflicting situation.
Notation "⟶": ignored transition

The total Σ-homomorphism between $SYNC_i$(MC) and $SYNC_j$(SC), is verified.

- $SYNC_i$ and $SYNC_j$ are Σ-algebras.
- figure 1 represents the commutations between $SYNC_i$ and $SYNC_j$. Condition (ii) of the theorem is satisfied.

It is important to note that the transitions indicated on figure 1 are the result of an exhaustive analysis of operator semantics described in DLP specifications.

We can notice the verification of condition (i) of the theorem. Canonical representation of T_i and T_k are respectively :

$PREPAREG_i$-$MANIPULEG_i$-$TERMINEG_i$ (T_i is processed)

$PREPAREG_k$-$TERMINEG_k$

and (T_k is temporarily rejected)

$PREPAREG'_k$-$TERMINEG'_k$

Notes on figure 1

(1) - We reach homogeneous states depicted in case (i).

(2) - We introduce indices to DLP operators and to characterize these attached to T_i-T_k

(3) - A level change corresponds to a change of the transaction processed by a controller

(iii) unreliable environment

If the MC crash occurs during the first step of synchronization we get the following canonical representation :

TERMINE(PREPARE(DBD,O_VB),MSD,L))

We are going to concentrate our proof on the critical case where the MC crash occurs during the second step of synchronization (one update at least has been performed).

We distinguish two major cases corresponding to a MC crash during the broascasting of

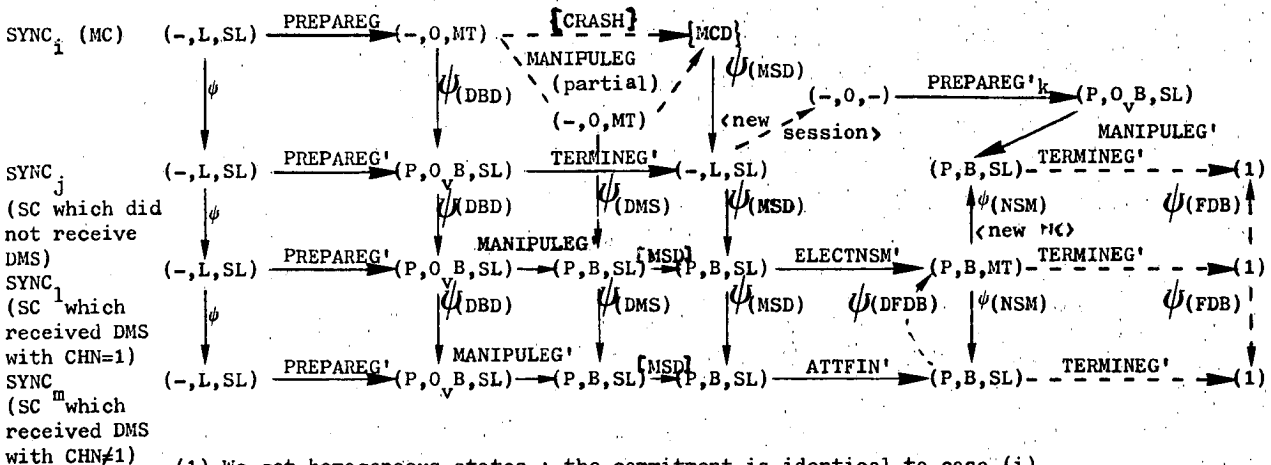
- the DMS (or NSM) message.
- the FDB message.

1 - MC crash during the DMS broadcasting phase

In DLP we do not have a global knowledge of the remote states attached to the duplicated entity. Therefore we may face the following situation : some slave controllers have their local copy blocked (B state), are waiting for the DMS, notice the failure of the MC and give up the synchronization session ; they are in their first step of synchronization. Other slave controllers received the DMS and processed it ; they are in their second step of synchronization. In this case the MC crash occurs during the DMS-broadcasting phase.

Figure 2 visualizes this situation. We represent the three possible types of SC :

- the SC which did not receive the DMS ($SYNC_j$)
- the SC which received the DMS with CHN = 1 ($SYNC_1$)
- the SC which received the DMS with CHN ≠ 1 ($SYNC_m$)



(1) We get homogeneous states ; the commitment is identical to case (i)

[M] Operator associated with the reception of the message M.

Figure 2 : Commutation diagram corresponding to a MC crash during the DMS (or NSM)-broadcasting phase.

The election of a new MC ($SYNC_1$) is performed.
Both conditions of the theorem are satisfied.

2 - MC crash during the FDB broadcasting phase

If we consider a crash during this phase.
we get a critical situation where some controllers finished up the session while others are still waiting for the FDB.

Such a situation is depicted in figure 3.

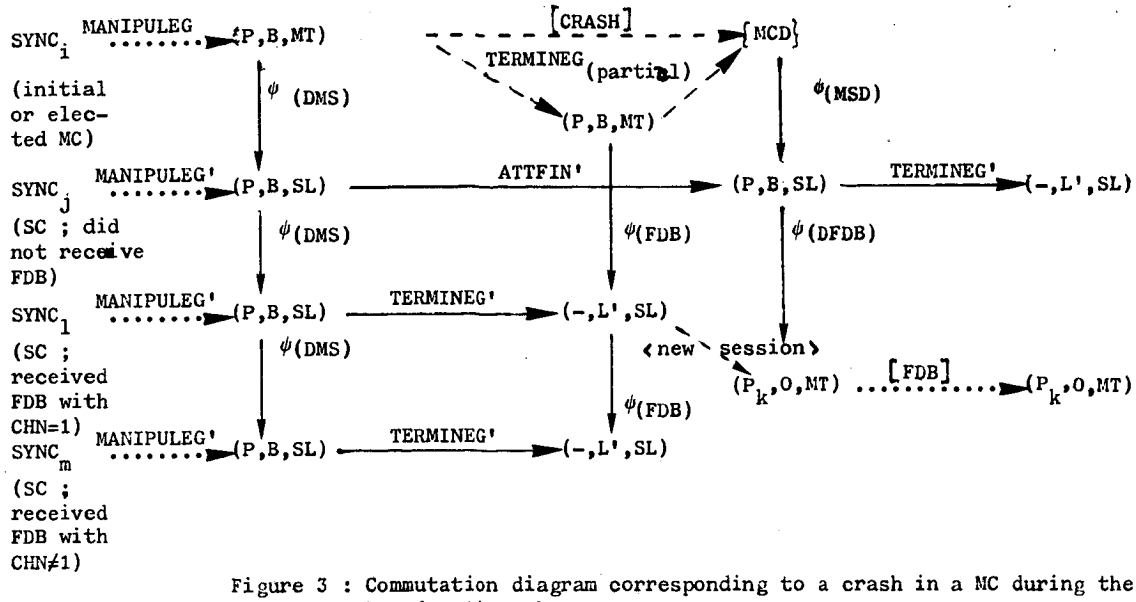


Figure 3 : Commutation diagram corresponding to a crash in a MC during the FDB broadcasting phase.

Notes : (i) In this figure $SYNC_1$ did not receive the FDB ; we are in the previous case with a commitment using a new MC.

(ii) In the case of robust synchronization states, the canonical representation on each active controller is PREPARE-MANIPULE-TERMINE (a transition like ATTFIN' is only a temporary step which does not affect the local state of a copy).

As a conclusion of this proof, let us say that DLP ensures strong mutual integrity of a replicated data base in any environment.

Assertion 2

DLP ensures strong mutual integrity of modification journals

Proof

First of all, let us assume that every controller which received a CTACT table, correctly built modification journals for failed sites.

The proof is reduced to show that

(i) CTACT table is correctly updated.

(ii) every active controller gets the same CTACT table when synchronization session is over.

Point (ii) is shown with the previous commutation diagrams which visualize a global session since every controller which receives the FDB, receives the correct CTACT table (contained in every broadcasted message).

Let us consider point (i) and the general following situation ; the MC (initial or elected) notes a SC crash after the broascasting of the DBD or DMS (or NSM). This situation may be represented by the functional correspondance :

$$(P,O_V B,SL) \xrightarrow{[SSD]} (P,O_V B,MT)$$

Assertion 1 shows that, when there exists a MC crash, DLP returns to a situation with a new MC and cooperating SC's.

Operators semantics corresponding to MSD reception by an SC, or SSD reception by a MC, show that the CTACT table is updated before its inclusion in the messages DMS, NSM and FDB broadcasted by the MC.

Therefore, each active controller receives an up-to-date CTACT.

Modification journals are then identical at the end of a synchronization session.

As a summary let us say that the two assertions indicated here correspond to the two aspects of DLP fault-tolerance :

- existence of a virtual ring of resiliency to elect a new MC.
- existence of a fully-replicated modification journal to recover from crash.

Conclusion

Algebraically-defined data types represent a formal tool conducive to clear specifications, simple verifications (proof of correctness of the initial objectives of the formalized system) and automatic implementation[20].

However there exist some weak points associated with validation (proof of correctness of the specifications) of algebraically-defined data types. There are two undecidable problems :

- the problem of consistency which concerns the avoidance of contradictions among the axioms.
- the problem of completeness which concerns the complete definition of the operations.

We can exhibit sufficient conditions for consistency and completeness and check whether they are satisfied. As a consequence, a specification approach is to start from valid ADT's and obtain the others we are interested in, by applying "type constructors"[7] or "enrichment"[4].

Automatic validation tools are being designed[16]... They are based on automatic mappings to other formal models (first order logic,...) where automated tools exist (PROLOG,...). This approach is also developed in[2].

In the realm of (synchronization) protocols, we can give an informal definition of specification completeness taken from[26] : a protocol description is said to be "complete" if the behaviour of every entity manipuled in the protocol is defined in every possible situation ; i.e. if the semantics associated with the reception of any synchronization message in any situation, is defined.

In this sense, DLP specifications are complete.

If we now consider specification consistency, we may say that protocol specifications are consistent if every time a message is generated by an operator, there exists another "symmetric" operator having this message as input.

In this sense, DLP specifications are consistent.

Acknowledgments

Special thanks are due to G. POPEK for helpful comments regarding this approach.

Referenoes

1. ELLIS C.A. "Consistency and correctness of dupli cated data base systems". Proc. 6th Symp.on operating systems principles,Purdue Univ.Nov.77,pp. 67-84.
2. Van EMDEN M.H.,MAIBUM T.S."Equations compared with clauses for specification of abstract data types".Workshop on formal models for data bases. CERT-DERI, Toulouse,France,Dec.1979.
3. GERHART S.L.et al"An overview of AFFIRM : a specification and verification systems" USC research report,Nov.26,1979.(also Proc.of IFIP Congress 1980,pp 343-349)
4. GOGUEN J.A. et al "Initial algebras,semantics and continous algebras".JACM 24,1977,pp.66-95.
5. GOGUEN J.A.,TARDO J."An introduction to OBJ:a language for writing and testing formal algebraic program specifications".Proc.Conf.on specif. of reliable software.Cambridge,MA,USA,April 1979 pp. 170-189.
6. GRAY J.N."Notes on data base operating systems" Lecture Notes in Comp.Science,edit.by Goos and J.Hartmanis,Springer Verlag,Berlin,New York, 1978,pp.394-481.
7. GUTTAG J.V. et al"Abstract data types and software validation".CACM Dec.78,Vol 25,n°12, pp. 1048-1064.
8. LE LANN G."Algorithms for distributed data sharing systems which use tickets".SIRIUS research report,SYNC 1003,1978(Proc.3rd Berkeley Workshop San Francisco,August 1978,pp.259-272)
9. MIRANDA S."Système d'accès interactif à des fichiers et copies de fichier sur un réseau d'ordinateurs".Thèse de 3ème Cycle,Univ.Paul Sabatier, Toulouse,France,Nov.1976.
10. MIRANDA S."Specification and validation of two ring-structured synchronization protocols for distributed data bases".Proc.Int.Symp. on distributed data bases.Paris,March 12-14,1980,pp.347-275
11. MIRANDA S."Architecture formelle d'un système d'intégrité pour une base de données réparties". Thèse d'Etat,Univ.Paul Sabatier,Toulouse,France, Sept.1980.
12. MIRANDA S."DLP : a fault-tolerant decentralized locking protocol for distributed data bases".Proc FTCS 10,Oct.1-3,1980,Kyoto,Japan,pp.83-87.
13. MIRANDA S."Performance considerations of DLP, a decentralized locking protocol for distributed data bases".Proc.3rd Hungarian CSC,Budapest,Jan. 26-28,1981, pp 197-210.
14. MIRANDA S."Abstract data types as a formal tool for system design".Proc.1981 CISS,Shaffer Hall, John Hopkins Univ.Baltimore,March 25-27,1981.
15. MIRANDA S."A formal model for a synchronization protocol based on a primary copy technique".Proc 12th Annual Pittsburgh Modeling and Simulation Conference,Univ.of Pittsburgh,April 30 -May 1, 1981.
16. NOURANI F."Constructive extension and implementation of abstract data types and algorithms". PH-D Thesis,UCLA,UCLA-ENG 7945,August 1979.
17. OKADA K. et al"Reliable program derivation in functional language by applying JACKSON's design method".Proc.FTCS 10,Kyoto,Japan,Oct.1980,pp. 91-96.
18. PARDO R.,LIU M.T.,BABIC G.A."Distributed services in computer networks : designing the distributed loop data system (DLDBS)". Research report Ohio State Univ. 1980.
19. REICHEL H."Behavioural equivalence : a unifying concept for initial and final specification methods".Proc.3rd Hungarian CSC,Budapest,January 26-28,1981,pp.235-246.
20. TARDO J."OBJ and the automatic implementation of abstract data types".PH-D Thesis,UCLA 1977.
21. TARDO J. et al "A practical method for testing algebraic specifications".UCLA Quarterly,January 1979,Vol 7,n° 1,pp.68-95.
22. THATCHER J.N."Specification of abstract data types using conditional axioms".IBM Research report,RC 6214,Sept.1976.
23. THOMAS R.H."A solution to the update problem for multiple copy data bases which use distributed control". BBN report,n°3340,July,1975.
24. UCLA Computer Science Department Quarterly, Research and Education, Fall Quarter 1980,vol 8,n° 4.
25. WILMS P. et al "A majority consensus algorithm for the consistency of duplicated and distributed data".Proc.1st International Conference on distributed computing systems, Huntsville,Alabama, Oct.1-5, 1979.
26. ZAFIRULO P."Protocol validation by duologue-matrix analysis". Proc.International Communication Conference,Chicago,June 1977, pp.359-363. (IEEE Transaction on Communication A6, August 1978).